Listas en Python: operaciones básicas

Yan Orestes
01/10/2020

Una profesora de matemáticas desarrolló una técnica de evaluación diferente, en un intento de aumentar la interacción entre los estudiantes para la próxima prueba, la calificación de todos sería la misma, equivalente al promedio de todas las calificaciones.

Todo este proceso lo haría un sistema **Python** automático. Las calificaciones de los estudiantes en la sala evaluada se archivaron en una estructura de **lista**:

```
notas = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7
```

La fórmula para calcular la media aritmética de una lista de números es la **suma de los números dividida por la cantidad de números**. Luego necesitamos descubrir dos cosas: la **suma de las notas** y el **número de notas**.

Sumar todos los elementos de una lista

Para sumar todos los elementos de la lista, comenzaremos inicializando nuestra variable que contendrá el resultado de la suma de las notas con el valor 0, lo que indica que la suma aún no se ha realizado:

```
suma_de_notas = 0
```

A partir de ahí, un enfoque que podemos seguir es a través de una iteración sobre la lista de notas. Es decir para cada nota, su valor se agrega a la variable `suma_de_notas`, de la siguiente manera:

```
for nota in notas:
    suma_de_notas  += nota

print(suma_de_notas)
```

Y el resultado:

```
98.0
```

Funciona perfectamente! Pero, ¿no hay una forma más sencilla de sumar todos los elementos a una lista?

Sumar los elementos de una lista con la función sum ()

Sumar todos los elementos de un conjunto es una tarea muy común en la programación. Por lo tanto, Python también tiene una función nativa dedicada a calcular la suma de todos los elementos en una lista: ¡la función sum ()!

Usemos este nuevo enfoque:

```
notas = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7

## código

suma_de_notas = sum(notas)
print(suma_de_notas)
```

Y así:

```
98.0
```

No solo es más simple usar la función `sum()` que implementar un algoritmo de suma a través de un bucle, sino que su rendimiento es mucho mejor: un loop `for` se ejecuta como el ***bytecode*** interpretado de Python, mientras que la función `sum()` se escribe exclusivamente en el lenguaje C.

Ahora que ya conocemos la suma de las notas, necesitamos averiguar cuántas son.

Averiguar el tamaño de una lista con la función len()

El número de notas es el tamaño de la lista. Podemos, por supuesto, nosotros mismos contar y almacenar ese recuento en una variable:

```
cantidad_notas = 15
```

Bien, pero este enfoque no sería la mejor opción caso la lista cambie de tamaño, o incluso si la lista es más grande (¡imagínate contando número por número en una lista de más de 100 elementos!)

Si ya hemos trabajado con otros lenguajes de programación, probablemente hayamos visto que generalmente hay una forma nativa más simple de calcular el tamaño de una lista. ¡Python no es la excepción!

La función *built-in* len() devuelve la longitud de un objeto cuya clase implementa el método len(), en nuestro caso en una lista. Probémoslo con nuestras notas:

```
notas = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7

#código

cantidad_notas = len(notas)
print(cantidad_notas)
```

El resultado:

```
15
```

Calcular la media aritmética

Ahora que tenemos los dos valores que necesitábamos, calculemos el promedio usando la fórmula `sum_de_los_numeros` / `cantidad_de_numeros`:

```
nota_media = suma_de_notas / cantidad_notas
print (nota_media)
```

Ahora tenemos el promedio:

```
6.5333
```

El resultado fue revelado a los estudiantes, quienes respondieron con quejas: la calificación fue más baja que lo esperado.

Después de analizar las calificaciones, descubrimos que el promedio fue calculado dos notas **0** de dos estudiantes que estuvieron ausentes el día del examen, ¡lo cual no era justo!.

Por lo tanto, debemos eliminar estas notas con valor **0** de nuestra lista y volver a calcular el promedio.

Eliminar un elemento de una lista con el método remove()

Ahora que tenemos nuestra lista completa, ¿cómo podemos eliminar un elemento de ella?. Para esto, Python nos trae el método `remove()`, que toma como parámetro el **valor** que queremos eliminar de nuestra lista. Vamos a probar:

```
notas = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7

notas.remove(0)
print(notas)
```

Y el resultado:

```
[0, 9.0, 8.0, 5.0, 10.0, 7.0, 0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7.5]
```

Logramos eliminar uno de los **0**, pero el otro aún permaneció. ¿Porque sera? Resulta que el método `remove()` elimina solo el primer elemento encontrado cuyo valor es el mismo que el parámetro pasado, no todos los elementos con ese valor.

¿Necesitamos entonces llamar al método `remove()` dos veces? Bien, eso funcionaría, pero ¿y si, en lugar de dos, tuviéramos diez notas **0**?. ¿Realmente tendríamos que repetir la misma línea de código diez veces? Esto no parece ideal.

Conociendo Python y la lógica de programación, un enfoque intuitivo para este problema sería iterar sobre la lista completa con un bucle for y eliminar el valor si es igual a **0**, para que no tengamos que preocuparnos por la cantindad de notas **0**. Probemos:

```
notas = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7

for nota in notas:
    if nota == 0:
        notas.remove(nota)

print(notas)
```

E o resultado:

```
[0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7.5]
```

¡De nuevo queda un **0**! ¿Por qué esta vez? En este caso, lo que sucedió fue una confusión entre el bucle for y el método `remove()`.

El for itera sobre los elementos de una lista a través del índice, es decir, comienza con el elemento en la posición 0, se mueve al elemento en la posición 1 y así sucesivamente, hasta que el índice (la posición) alcanza la longitud de la lista.

El método `remove()`, a su vez, al eliminar un elemento, reasigna los demás dentro de la lista para que no haya espacio vacío en el lugar donde se eliminó el elemento. Cuando eliminamos un elemento de la lista, la lista en realidad se reduce de tamaño.

Sabiendo esto, hagamos una simulación paso a paso para tratar de entender lo que sucedió:

PRIMER LOOP: índice = **0** lista = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7.5]

SEGUNDO LOOP: índice = **1** lista = [0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7.5]

Tenga en cuenta que el segundo 0 fue ignorado por el bucle `for`. Cuando el `for` llegó al índice `[1]` con valor 0 inicialmente, ese 0 ya se había recorrido una posición y se había reasignado al índice `[0]`, lo que resultó en este problema.

Por lo tanto, siempre debemos recordar que se recomienda **no alterar directamente nunca una lista mientras se itera sobre ella.**

Necesitamos una mejor manera de eliminar todas las ocurrencias del mismo valor en una lista.

Filtrar una lista usando *list comprehensions*

Hay una técnica en Python llamada comprensión de listas. ¿Cómo podemos usarlo para resolver nuestro problema?. Queremos una lista de todos los valores en nuestra lista de calificaciones, siempre que el valor sea mayor que `0`.

Entonces:

```
notas = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7

notas_validas = [nota for nota in notas if nota > 0]
print(notas_validas)
```

Y así:

```
[9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7.5]
```

Finalmente, es solo recalcular el promedio con esta nueva lista de notas:

```
media_valida = sum (valid_notes) / len (valid_notes)
print (media_valida)
```

Esta vez, obtuvimos un mejor promedio para los estudiantes:

```
7.538461538461538
```

Después de calcular el promedio y redondearlo a 7.5, la maestra quería verificar si su método de evaluación había sido efectivo, es decir, si la calificación en el aula realmente representaba el desempeño de los estudiantes en la prueba.

Para hacer esto, necesitaba calcular cuántos estudiantes obtuvieron un puntaje cerca del promedio, con una diferencia de **0.5** (es decir, cuántos estudiantes obtuvieron **entre 7 y 8**).

Contar elementos en una lista con el método `count()`

Para contar cuántas veces aparece un elemento en una lista, podemos usar el método `count()` de esta manera:

```
cantidad_sietes = notas.count(7.0)
print (cantidad_sietes)
```

El resultado:

```
3
```

Para realmente hacer lo que queremos, necesitamos usar el método con tres valores: **7, 7.5 y 8**:

```
cantidad_alumnos_media = notas.count(7.0) + notas.count(7.5) + notas.count(8.0
print (cantidad_alumnos_media)
```

El resultado:

```
8
```

¡Correcto! En nuestro caso, esta solución con `count()` funciona bien, porque sabemos que solo queremos estos tres valores (`7.0, 7.5 y 8.0`).

Pero, ¿qué pasaría si nuestra lista tuviera, por ejemplo, valores como **7.1, 7.2** ... y así sucesivamente hasta **8.0**. ¿Tendríamos que hacer un `count()` para cada uno de estos valores? Parece complicado, ¿no?.

Otro enfoque que podría usarse, que evitaría el problema de tener más valores en el rango que queremos contar, sería usar un bucle for y almacenar el valor de conteo en una variable, como vemos a seguir:

```
notas = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0, 7

cantidad_alumnos_media = 0
for notas in notas:
    if 7.0 <= nota <= 8.0:
        cantidad_alumnos_media +=1

print (cantidad_alumnos_media)
```

Tenemos el mismo resultado:

```
8
```

Al consolidar lo que aprendimos del método `sum()` y de la comprensión de listas, y sabiendo que en Python la clase bool, que representa los tipos booleanos, hereda de la clase int (**True es igual a 1 y False es igual a 0**), podemos incluso aplicar esta lógica de una manera mucho más resumida:

```
grados = [0, 0, 9.0, 8.0, 5.0, 10.0, 7.0, 7.5, 4.0, 10.0, 7.0, 7.0, 8.0, 8.0,

cantidad_alumnos_media = sum ([7.0 <= nota <= 8.0 for nota in notas])
print (cantidad_alumnos_media)
```

Tenemos el mismo resultado:

```
8
```

Conclusión

En esta publicación pudimos ver la utilidad de las listas, cuándo podemos usarlas y qué podemos hacer con ellas.

Pasamos por necesidades como eliminar todas las ocurrencias de un valor en la lista, agregar todos los elementos en la lista, conocer la longitud de la lista e incluso eliminar todas las ocurrencias de un elemento en la lista, y logramos resolverlos uno por uno a través de nuestra lógica y utilizando las características propias de Python.

¿Qué tal aprender más sobre **Python** y sus diversos recursos para Ciencia de Datos?. ¡Mira nuestros cursos de **Python para Data Science** aquí en Alura!

ARTÍCULOS DE TECNOLOGÍA > DATA SCIENCE

Nuevos cursos
cada semana

NAVEGACIÓN

PLANES
INSTRUCTORES
BLOG
POLÍTICA DE PRIVACIDAD
TÉRMINOS DE USO
SOBRE NOSOTROS
PREGUNTAS FRECUENTES

¡CONTÁCTANOS!

¡QUIERO ENTRAR EN CONTACTO!

BLOG

PROGRAMACIÓN
FRONT END
DATA SCIENCE
INNOVACIÓN Y GESTIÓN
DEVOPS

AOVS Sistemas de Informática S.A
CNPJ 05.555.382/0001-33

SÍGUENOS EN NUESTRAS REDES SOCIALES

ALIADOS

Empresa participante do SCALE UP ENDEAVOR